SCKAN-Compare

Shailesh Appukuttan, Pranjal Garg, Hiba Ben Aribi, Gautam Kuma

Aug 31, 2023

CONTENTS

A package for retrieving and visualizing data contained in SCKAN (e.g., across species, relationship to spinal segments) to highlight similarities and differences in neuronal pathways

CHAPTER

ONE

INTRODUCTION

The body's involuntary physiologic processes is regulated by three distinct systems: sympathetic, parasympathetic and enteric, collectively called the autonomic nervous system. Any deviation from the optimal functioning of the system results in acute or chronic disease conditions. In developing suitable interventions/therapy to tackle the disease condition, clinicians need to be able to compare similarities/ differences between model organisms and humans. To achieve this goal, SKCAN-Compare uses the SPARC data. SKCAN-Compare can retrieve and visualise data between two organisms and compare the functional connectivity between the two organisms, thereby decreasing the time and resources spent on laboratory studies and assisting in clinical studies.

CHAPTER

TWO

SKCAN-COMPARE

SKCAN-Compare offers its functionality in two forms:

1. Python package
2. Web app

The Python package can be installed via PyPI. It allows users to customize and employ the took in their own workflows.

The Web App offers a ready-to-use tool to help compare SCKAN data. The app internally uses the Python module.

CHAPTER

THREE

CONTENTS

3.1 Introduction

3.1.1 Installation

If you are relying on Python packages for several, independent projects, we recommend that you make use of separate environments for each project. In this way, you can safely update and install packages for one of your projects without affecting the others. Both, `conda` and `pip` support installation in environments – for more explanations see the respective instructions below.

Standard install

We recommend installing into a separate "virtual environment", see the Python Packaging User Guide for more information. SCKAN-Compare is included in the PyPI package index: https://pypi.org/project/SCKAN-compare/ You can therefore install it with the `pip` utility:

```
pip install sckan-compare
```

In rare cases where your current environment does not have access to the `pip` utility, you first have to install `pip` via:

```
python -m ensurepip
```

Development install

If you wish to contribute to the development of SCKAN-Compare, the easiest way of setting up your environment would be as follows:

```
git clone https://github.com/SPARC-FAIR-Codeathon/2023-team-4
cd 2023-team-4
pip install -e .
```

Installing with the -e flag will allow changes to the source files to be immediately reflected in your environment.

3.1.2 Basic Usage

The functionality of the Python module can be accessed via the following classes:

1. SckanCompare
2. Visualizer
3. SimpleVisualizer

Detailed examples are available via the Jupyter notebook tutorials. Here, we show a few example code snippets of their usage:

You can import each of the classes via:

```
from sckan_compare import SckanCompare
from sckan_compare.visualize import Visualizer
from sckan_compare.simplevis import SimpleVisualizer
```

You can execute a SPARQL quey via:

```
result = sc.execute_query(<<query string>>)
```

Simple block plot can be created via:

```
simvis = SimpleVisualizer()
fig = simvis.plot_figure(result_df, selected_Region_A, selected_Region_B)
```

3.1.3 Web App

The web app is a Python-based app developed using 'Shiny for Python'. Shiny makes it easy to build web applications with Python code. It enables you to customize the layout and style of your application and dynamically respond to events, such as a button press, or dropdown selection. The examples on this site are rendered in the browser using Pyodide, but you can also install shiny to use it with your own projects.

We leveraged this functionality to provide a GUI interface to users to readily access the features of SCKAN-Compare. The app can also be run locally. You will need to install some packages for this. These can be installed as follows:

```
pip install shiny
pip install --upgrade shiny htmltools
pip install shinyswatch
pip install shinywidgets
```

Once the environrment is setup, the app can be launched via:

```
shiny run --reload
```

The above command should be run in the directory containing the file *app.py*, which contains the source code for the app.

3.1.4 Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at team@briansimulator.org. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

3.2 Code Docs

3.2.1 SckanCompare Class

A package for retrieving and visualizing data contained in SCKAN (e.g., across species, relationship to spinal segments) to highlight similarities and differences in neuronal pathways

License: Apache License 2.0

class sckan_compare.**SckanCompare**(*endpoint='https://blazegraph.scicrunch.io/blazegraph/sparql'*, *max_cache_days=7*)

Bases: `object`

Base class for accessing functionality

execute_query(*query_string*, *species=None*, *cached=True*)

Execute a SPARQL query and return the result.

Parameters

- **query_string** (*str*) – The SPARQL query string to execute.
- **species** (*str, optional*) – The species to consider in the query, if applicable.
- **cached** (*bool, optional*) – Whether to use cached data if available. Defaults to False.

Returns
The query result.

Return type
list

get_filtered_dataframe(*result*, *species=None*, *filter_column=None*, *filter_value=None*)

Create a filtered DataFrame from a query result. Replaces all synonyms for species and regions with unique labels, followed by the deletion of duplicate rows.

Parameters

- **data** (*list*) – The query result.
- **species** (*str*) – The species for which the data is provided.

- **filter_column** (*str, optional*) – The column to be filtered.
- **filter_value** (*str, optional*) – The value to be used for filtering column.

Returns
The filtered DataFrame.

Return type
pandas.DataFrame

get_valid_phenotypes()

Retrieve a list of valid phenotypes from the data source.

Returns
Dict with valid phenotypes URIs as keys and corresponding labels as values.

Return type
dict

get_valid_phenotypes_circuit_role()

Retrieve a list of valid phenotypes with circuit role from the data source.

Returns
Dict with valid phenotypes URIs as keys and corresponding labels as values.

Return type
dict

get_valid_regions_specify_species(*species*, *region=None*)

Retrieve a list of valid regions for a specific species from the data source.

Parameters

- **species** (*str*) – The species for which to retrieve valid regions.
- **region** (*str, optional*) – The region for which to retrieve valid regions. Valid values are: A, B, C or None. Defaults to None and returns all regions.

Returns
Dict of valid region URIs as keys and corresponding labels as values

Return type
dict

get_valid_species()

Retrieve a list of valid species from the data source.

Returns
Dict with valid species URIs as keys and corresponding labels as values.

Return type
dict

plot_dataframe_anatomy_vis(*df*, *species=None*, *region_A=None*, *region_B=None*, *region_C=None*)

Plot anatomical connectivity map based on a DataFrame.

Parameters

- **df** (*pandas.DataFrame*) – The DataFrame containing connectivity information.
- **species** (*str, optional*) – The species for visualization.
- **region_A** (*str, optional*) – The source region for filtering.
- **region_B** (*str, optional*) – The target region for filtering.

- **region_C** (*str, optional*) – The intermediate region for filtering.

Returns
The Plotly figure widget.

Return type
go.FigureWidget

plot_dataframe_block_vis(*df*, *region_A=None*, *region_B=None*)

Plot anatomical connectivity map based on a DataFrame.

Parameters

- **df** (*pandas.DataFrame*) – The DataFrame containing connectivity information.
- **species** (*str, optional*) – The species for visualization.
- **region_A** (*str, optional*) – The source region for filtering.
- **region_B** (*str, optional*) – The target region for filtering.

Returns
The Plotly figure widget.

Return type
go.FigureWidget

replace_region_synonyms_dataframe(*df*, *species*)

Replace region synonyms in a DataFrame with unique labels.

e.g. 'ovary' : http://purl.obolibrary.org/obo/UBERON_0000992 has several synonyms, such as 'animal ovary', 'female gonad', etc. This method is used to map these synonyms to the parent label.

Parameters
df (*pandas.DataFrame*) – The DataFrame containing region information.

Returns
The DataFrame with replaced region synonyms.

Return type
pandas.DataFrame

replace_species_synonyms_dataframe(*df*)

Replace species synonyms in a DataFrame with unique labels.

e.g. 'Rattus norvegicus' : http://purl.obolibrary.org/obo/NCBITaxon_10116 has several synonyms, such as 'brown rat', 'Norway rat', 'rats', 'rat'. This method is used to map these synonyms to the parent label.

Parameters
df (*pandas.DataFrame*) – The DataFrame containing species information.

Returns
The DataFrame with replaced species synonyms.

Return type
pandas.DataFrame

3.2.2 AntomyVis Class

A class for creating interactive brain region visualizations using Plotly.

param region_dict
Dictionary mapping region names to (x, y) coordinates.

type region_dict
dict

param species
The species for which the visualization is being created.

type species
str

`sckan_compare.anatomyvis.AntomyVis.region_dict`
Dictionary mapping region names to (x, y) coordinates.

Type
dict

`sckan_compare.anatomyvis.AntomyVis.species`
The species for which the visualization is being created.

Type
str

`sckan_compare.anatomyvis.AntomyVis.SCALE`
Scaling factor for the visualization.

Type
int

`sckan_compare.anatomyvis.AntomyVis.MAX_X`
Maximum X coordinate.

Type
int

`sckan_compare.anatomyvis.AntomyVis.MAX_Y`
Maximum Y coordinate.

Type
int

`sckan_compare.anatomyvis.AntomyVis.NODE_RADIUS`
Radius of the node markers.

Type
float

`sckan_compare.anatomyvis.AntomyVis.fig`
Plotly figure widget for visualization.

Type
go.FigureWidget

`__init__(region_dict, species):`
Initialize the AntomyVis class.

get_json_species_map(species=None):

Load a JSON species mapping of region names to (x, y) coordinates for anatomical visualization.

draw_rect(start_x, start_y, width=1, height=1, color_border="#4051BF", color_fill="#C5CAE9", tooltiptext="<set name>"):

Draw a rectangular region on the visualization.

draw_poly(xlist, ylist, color_border="#4051BF", color_fill="#C5CAE9", tooltiptext="<set name>"):

Draw a polygonal region on the visualization.

draw_background():

Draw the anatomical background for species.

mark_node(region, color_border="#FF0000", color_fill="#FFFF00", small=False):

Mark a node (brain region) on the visualization.

interpolate_coordinates(point1, point2, resolution=0.1):

Interpolate between two cartesian coordinates using NumPy.

plot_dataframe(df):

Plot dataframe connectivity info.

draw_edge_AB(region1, region2, neuron=None):

Draw an edge (connection) between two nodes (regions) A and B.

draw_edge_ABC(region1, region2, region3, neuron=None):

Draw an edge (connection) between nodes (regions) A and B via C.

show_figure():

Display the Plotly figure widget.

get_figure():

Get the Plotly figure widget.

3.2.3 BlockVis Class

A class for creating block visualizations using Plotly.

param None

sckan_compare.blockvis.BlockVis.**SCALE**

Scaling factor for visualization.

Type
int

sckan_compare.blockvis.BlockVis.**MAX_Y**

Maximum Y coordinate.

Type
int

sckan_compare.blockvis.BlockVis.**MAX_X**

Maximum X coordinate.

Type
int

sckan_compare.blockvis.BlockVis.**icons**

Dictionary of icons for nodes.

Type
dict

sckan_compare.blockvis.BlockVis.**fig**

Plotly figure widget for visualization.

Type
go.FigureWidget

__init__():

Initialize the BlockVis class.

interpolate_coordinates(point1, point2, resolution=0.1):

Interpolate between two cartesian coordinates.

plot_figure(df, region_A, region_B):

Plot the connectivity block visualization

update_graph():

Update the layout of the figure.

draw_block_bg(x0, y0, x1, y1, opacity, color):

Draw a rectangular background block.

draw_image(icon, x, y):

Draw an image on the visualization.

draw_connections(x, y, label, color):

Draw connections between nodes.

add_text(x, y, text, fontsize=20):

Add text annotation to the visualization.

mark_node(x, y, label):

Mark a node on the visualization.

show_figure():

Display the Plotly figure widget.

get_figure():

Get the Plotly figure widget.

3.2.4 CacheManager Class

A class for managing a disk-based cache with expiration for cached data.

param cache_directory
Path to the directory for storing cache data.

type cache_directory
str

param max_cache_days
Maximum number of days a cached entry is considered valid.

type max_cache_days
int

sckan_compare.cachemanager.CacheManager.**cache**

Disk cache object for storing data.

Type
diskcache.Cache

sckan_compare.cachemanager.CacheManager.**max_cache_days**

Maximum number of days a cached entry is considered valid.

Type
int

__init__(cache_directory, max_cache_days):

Initialize the CacheManager class.

create_disk_cache(directory):

Create a disk cache object.

get_cached_data(key):

Retrieve cached data associated with a given key.

cache_data(key, data):

Cache data with an associated key.

invalidate_old_cache_entries():

Remove cached entries that have expired.

3.2.5 Utils Module

Utility methods for SckanCompare package.

License: Apache License 2.0

sckan_compare.utils.**filter_dataframe**(*df*, *column*, *value*)

Filter a DataFrame based on value in specific column.

Parameters

- **df** (*pandas.DataFrame*) – The DataFrame to be filtered.
- **column** (*str*) – The column to be filtered.
- **value** (*str*) – The value to be used for filtering column.

Returns
The filtered DataFrame.

Return type
pandas.DataFrame

sckan_compare.utils.**get_dataframe**(*data_as_list*)

Convert a list of data to a pandas DataFrame.

Parameters
data_as_list (*list*) – List of data to be converted.

Returns
The converted DataFrame.

Return type
pandas.DataFrame

sckan_compare.utils.**remove_duplicate_species**(*df*)

Replace various species synonyms with standard species names in a DataFrame.

Parameters
df (*pandas.DataFrame*) – The DataFrame containing species information.

Returns
The DataFrame with replaced species synonyms.

Return type
pandas.DataFrame

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX